Project Readme Template

Version 1 9/11/24

A single copy of this template should be filled out and submitted with each project submission, regardless of the number of students on the team. It should have the name readme_"teamname"

Also change the title of this template to "Project x Readme Team xxx"

1	Team Name: Shoe
2	Team members names and netids: Alex Schumann, aschuma4
3	Overall project attempted, with sub-projects: NTM Tracing
4	Overall success of the project: Successfully traces all tested NTMs, DTMs, and inputs as expected
5	Approximately total time (in hours) to complete: ~6
6	Link to github repository: https://github.com/CobblerOfShoes/NTM_Simulation
7	List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary)

File/folder Name	File Contents and Use
Code Files	
traceTM_Shoe.py	Driver program that takes in an NTM csv and list of strings and checks if the strings will be accepted The program uses the following flags: ● -f : takes in the filename of the TM csv file ● -s : takes in a list of strings to process ● -v : verbose, toggles printing of all configurations processed By default, the program prints out: ● The machine name ● The string being processed ● The result [accept, reject, max depth reached] ● The number of transitions simulated ● The average nondeterminism (average number of transitions per depth level) ● If accepted, the path taken to accept the string NOTE: The initial state is neither included in the number of transitions nor the average nondeterminism.
Test Files	

check_aplus.csv	NTM with known output from the project handout. Proves that NTM machines are processed as expected.
check_abastar_DTM.csv	DTM that proves that the program handles deterministic machines as expected, with one transition per level and an average determinism of 1.
check_loop.csv	NTM with some garbage transitions for the Regex pattern of (a U b)* to ensure that the program stops upon reaching max depth level.
abcstar.csv	Contains NTM description for the machine that accepts the regex pattern a*b*c*.
equal01s.csv	Contains NTM description for the machine that accepts { w | w has the same number of 0's and 1's}
equal01s_DTM.csv	Contains the DTM equivalent of equal01s.csv
Output Files	
output_aplus_Shoe.txt	Contains the results for checking the strings: "", "a", "aaa", "aaabb", "aaaaaaaaaaaaaaa" This was cross referenced with Prof. Kogge's handout to ensure that "aaa" was accepted as expected. All other strings processed as expected, where "" rejected, "a" accepted, "aaabb" rejected, and "aaaaaaaaaaaaaaa" accepted. With "aaa", there is an average non-determinism of 2.5. As the length of the input approaches infinity (seen with aaaaaaaaaaaaaaa), the average non-determinism approaches 3, which is the maximum number of future states available within a depth level. Therefore, the output makes sense.
output_abastarDTM_Shoe.txt	Contains the results for checking the strings: "a", "ab", "abaaa", "abaaaab" This saw an average non-determinism of 1 across all strings, which is expected for a DTM. The first three strings were accepted as expected, and the last was rejected as expected.
output_loop_Shoe.txt	Contains the results for checking the string "aabb" with a maximum depth level of 5. The program did stop upon reaching depth level 6 as expected, and it is known that the program would have never accepted.

	The max depth was set to be very low as there are a large number of transitions that allow for a very wide tree. It would take a very long time to simulate a max depth of 100, and since we know the TM will never accept, the max depth does not really matter much. The program simulated 77 transitions, with an average non-determinism of 15.40.
output_abcstar_Shoe.txt	Contains the results for checking the strings: “”, “abbb”, “abbbcaa”, “aaabbbbbccc” All input strings processed as expected, and the average non-determinism ranged around 7-7.33.
output_equal01s_Shoe.txt	Contains the results for checking the strings: “”, “011”, “0101”, “10001”. All strings processed as expected, and had an average non-determinism of around 1.5.
output_equal01s_DTM_Shoe.txt	Contains the results for checking the strings: “”, “0101”, “111”, “1001”. All strings processed as expected, and had an average non-determinism of 1 which is what the behavior should be for DTMs.
Plots (as needed)	
N/A	No plots necessary, all output is contained in output text files

8	Programming languages used, and associated libraries: Language: Python Libraries: argparse, copy (for deepcopy), os, csv, sys, collections (defaultdict and Counter)
9	Key data structures (for each sub-project): Tape: Represents the input string as a tape, with methods to edit the head and get the left and right of the head NTM_Transition: Holds transition information, which is the next state, the new character, and the direction to move in NTM_Configuration: Holds the current configuration for an NTM, with the current state, tape, head, and previous states.

	NTM: Holds the information for the Turing Machine, such as the transition function and the queue of available transitions
10	General operation of code (for each subproject) A command may look something like this: “python3 traceTM_Shoe.py -f check_loop.csv -s "aabb" -v -d 5 > output_loop_Shoe.txt” The arguments are: ● -f to select the NTM csv file description ● -s to pass a list of strings to test ● -v to toggle extra printing of all intermediate configurations tested ● -d to set the depth level The output can be redirected into a file, which was done to obtain the output files. Program flow: ● The program will read in the csv file to build the NTM ● The program will then iterate through the passed strings and test them on the NTM ● Starting from the initial state, the program will use a queue to grab the first state, search for all available transitions, and append them onto the queue. ● The program will index the transition function by the current state and input to get all possible transitions, and apply them to the current state to get all possible next configurations. ● This loop through the queue continues until an acceptance state is reached, no more states may be traveled to (all paths led to reject), or the maximum depth is reached.

11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. There are three check files with known outputs, which check the aplus NTM, the abastar DTM, and a dummy NTM that forces a loop. These three respectively prove successful behavior on NTMs, DTMs, and when the max depth is exceeded. Further, I used two more NTM files (equal01s and abcstar) and another DTM file (equal01s_DTM) to prove that the NTM and DTM behavior was consistent across multiple machines, and did not just work as a fluke.
12	How you managed the code development The code development was managed on GitHub. I began by discerning what data structures I needed, and then developed some pseudo code for my program flow.

	Off of this, I built the program which was rather easy as I now had a well defined problem.
13	Detailed discussion of results: The program functions as I expected it to. For DTMs, only one transition is taken per depth level, for an average non-determinism of 1. For NTMs, all possible future states are explored properly, and the program stops and prints its information as it should when an accept state is reached, when all paths lead to reject, and when the maximum depth level is reached. I have found that in order to get a good representation of the average non-determinism for an NTM, it is important to use longer strings as the average non-determinism tends to increase proportionally to the length of the string. Further, the upper limit for the average non-determinism is based on the structure of the NTM itself, and seems to have no larger pattern across NTMs.
14	How team was organized I did all the work as it was a solo project, so shoutout Alex Schumann
15	What you might do differently if you did the project again If done again, I may think more carefully about the time complexity of my program. I tried to get fancy with the classes I was using, but I think my print statements slow the code down a lot and can make longer strings take a long time to process. In the future I would research ways of trying to make print statements asynchronous with code execution.
16	Any additional material: I created a check_loop.csv NTM description that forces a loop for easy checking of if a max depth level is reached.